Keele University

KEELE UNIVERSITY

KEELE SCHOOL OF COMPUTER SCIENCE

SUBMISSION FOR SYSTEM DESIGN & PROGRAMMING REASSESSMENT TECHNICAL REPORT

MODULE NAME: SYSTEM DESIGN & PROGRAMMING

CODE: CSC-40044

STUDENT NUMBER: 20021044

TUTOR NAME: KP LAM

TABLE OF CONTENT

1.0 Building The Application

This document is drafted to guide the user through the design, implementation, and setup of the proposed application. The application will enable the user to pick from a list of destinations, view the shortest available route to get from the start point A to endpoint B, and enable them to indicate what paths(nodes) are blocked(inaccessible). The final application was built using Python and uses two distinct modules shipped with the basic python3 (Tkinter and math). After much research, the application was built using the A* search algorithm which is a very popular technique in path-finding and traversals (Liu, X. and Gong, D., 2011). The Ten Buildings selected from the Keele Campus Map (See figure 1) are MARRIOT HOTEL, BARNES HALL, IC1 (INNOVATION CENTRE), LINDSAY HALL, STUDENTS UNION, TAWNEY, CENTRE (CICC), VET SCHOOL, THE COVERT, JACK ASHLEY.

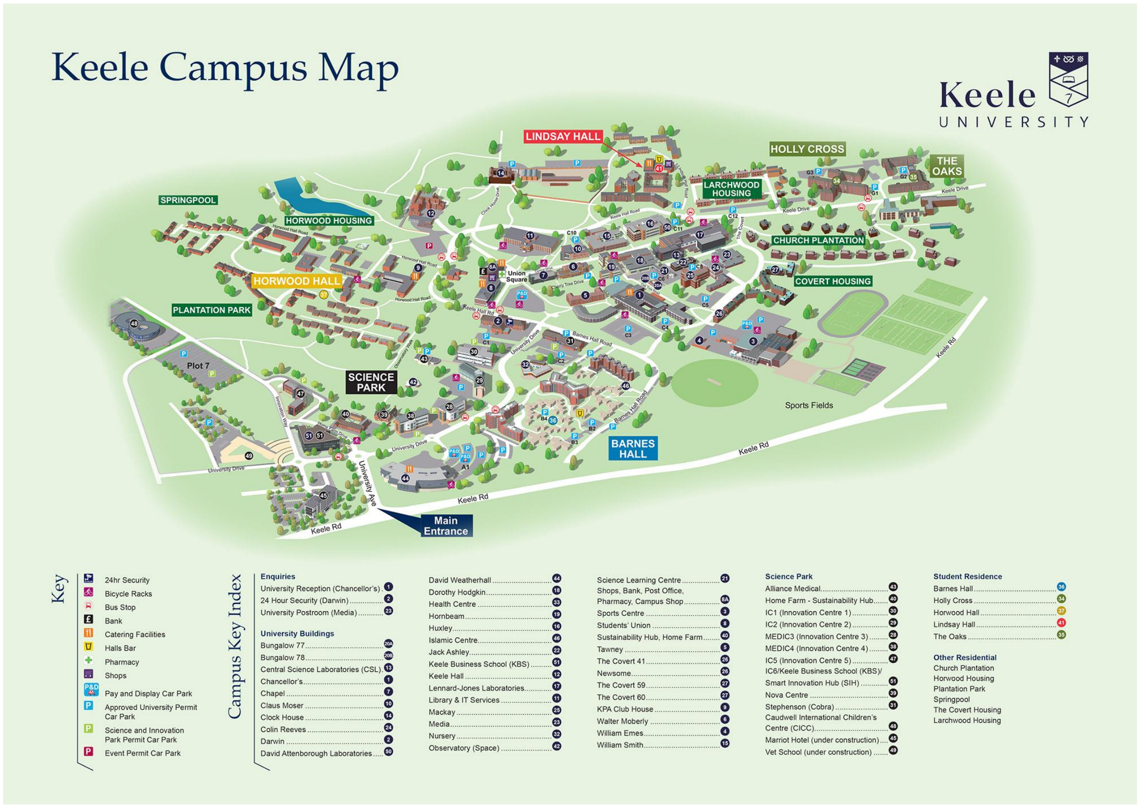


Figure 1:- Keele Campus Map

First, a twenty by twenty grid was drawn up to represent the user map for the application and each building was placed on specific nodes based on their location on the Keele Campus Map (See Figure 1). Taking into account the grid points listed below for each building (See Table 1), the application grid map was drafted (See Figure 2) while also considering all possible routes a user can take to move from one location to another.

Building Number on Grid & Keele Map	Building Name	Location on Map X-Axis	Location on Map Y-Axis
45	MARRIOT HOTEL	20	20
36	BARNES HALL	20	13
30	IC1(INNOVATION CENTER 1)	10	15
41	LINDSAY HALL	1	4
8	STUDENT'S UNION	6	11
5	TAWNEY	8	9
48	CENTRE (CICC)	3	20
49	VET SCHOOL	10	19
26	THE COVERT	20	5
22	JACK ASHLEY	13	7

Table 1:- Table showing building location on the drafted map in figure 2

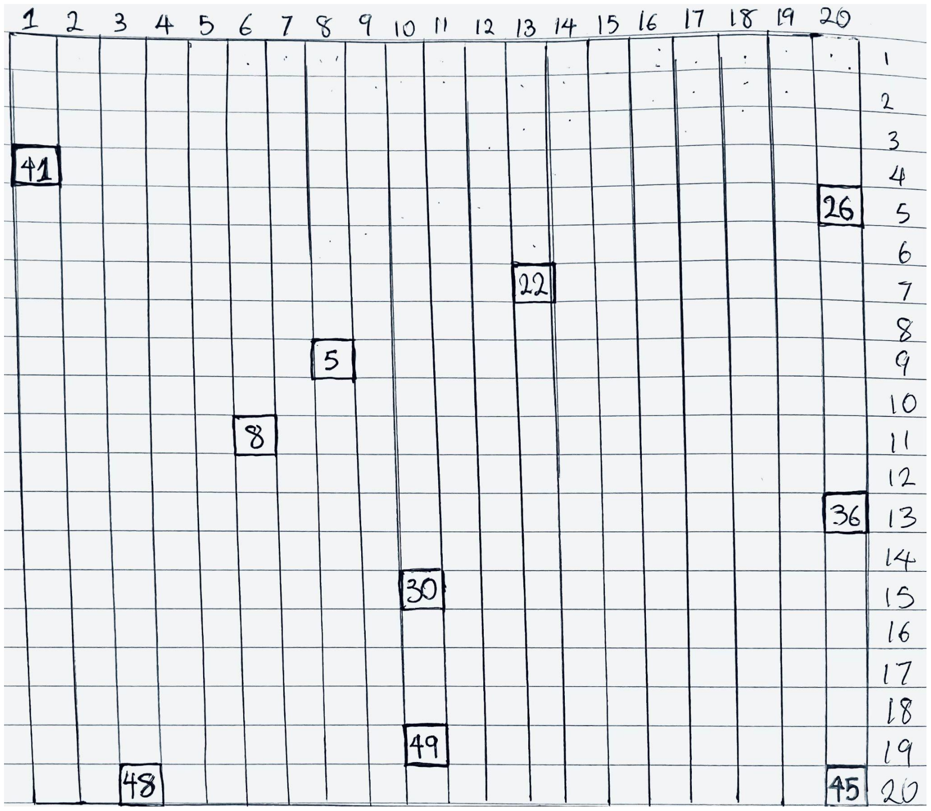


Figure 2:- Draft of the application map view

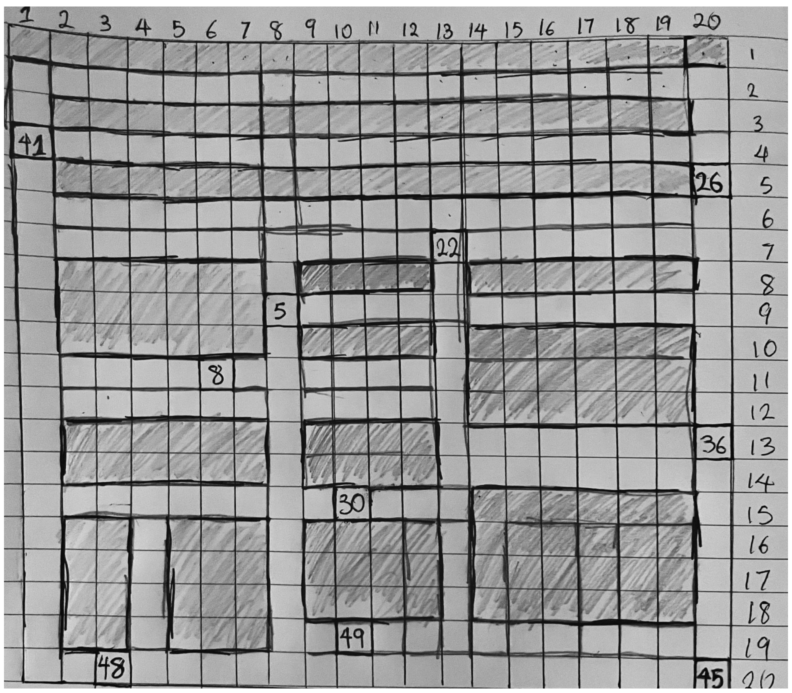


Figure 3:- Draft of the application map view with routing defined (Hashed paths indicate non-transversible areas)

2.0 A Star Search Algorithm

The motivation in using the A* algorithm over other algorithms lies in the fact that it is generally used to approximate the shortest path between nodes in a real-life situation like in games or maps where there can be a hindrance (blocked paths)(See Figure 3).

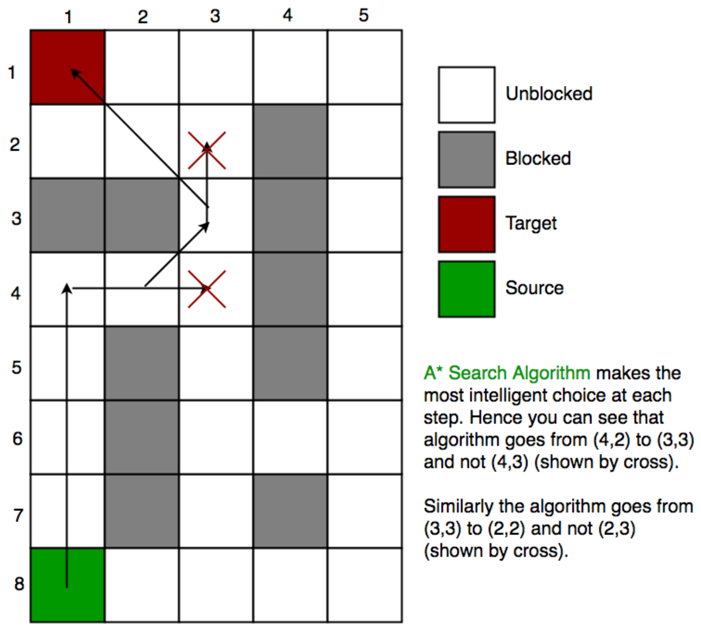

1 2 3 4 5
1 2 3 4 5 6 7 8
Unblocked
Blocked
Target
Source
A* Search Algorithm makes the most intelligent choice at each step. Hence you can see that algorithm goes from (4,2) to (3,3) and not (4,3) (shown by cross).
Similarly the algorithm goes from (3,3) to (2,2) and not (2,3) (shown by cross).

Figure 4:- Sample of A Search Algorithm*

A* Search algorithm is usually praised over other search algorithms as unlike other algorithms, it is referred to as having brains i.e it follows a set of rules and equations in order to get the shortest path possible in a route (Li, D.W., Han, B.M. and Han, Y., 2007). Note: it is worth mentioning that it has been used in many popular games and web-based maps to find the shortest path very efficiently (approximately).

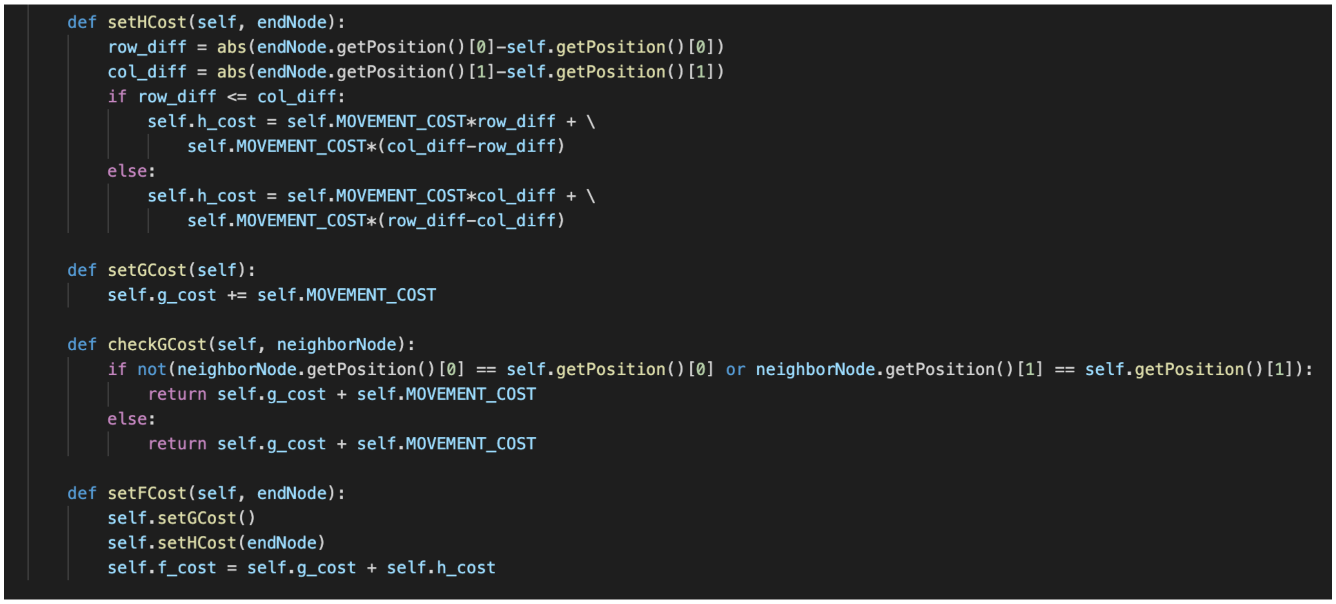
What A* Search Algorithm aims to do is that at each step it picks the node according to a value-'**f**' which is equal to the sum of two other parameters – g and h. At each step it picks the node having the lowest '**f**', and processes that node (See figure 4). We define g and h below:

g = the cost of movement required to move from the starting point to the current node on the grid.

h = the estimated cost of movement required to move from the current node on the grid to the final node. This is what is known as the heuristic.

In summary, we have:

- F - the total cost of node
- G - distance between current node and start node
- H - the heuristic - the estimated distance between the current node and the end node



```
    def setHCost(self, endNode):
        row_diff = abs(endNode.getPosition()[0]-self.getPosition()[0])
        col_diff = abs(endNode.getPosition()[1]-self.getPosition()[1])
        if row_diff <= col_diff:
            self.h_cost = self.MOVEMENT_COST*row_diff + \
                self.MOVEMENT_COST*(col_diff-row_diff)
        else:
            self.h_cost = self.MOVEMENT_COST*col_diff + \
                self.MOVEMENT_COST*(row_diff-col_diff)

    def setGCost(self):
        self.g_cost += self.MOVEMENT_COST

    def checkGCost(self, neighborNode):
        if not(neighborNode.getPosition()[0] == self.getPosition()[0] or neighborNode.getPosition()[1] == self.getPosition()[1]):
            return self.g_cost + self.MOVEMENT_COST
        else:
            return self.g_cost + self.MOVEMENT_COST

    def setFCost(self, endNode):
        self.setGCost()
        self.setHCost(endNode)
        self.f_cost = self.g_cost + self.h_cost
```

Figure 5:- Screenshot of equation showing how the F, G, and H is set

3.0 Setting Up And Running The Application

The main library to be concerned about is the python library (Van Rossum, G. and Drake Jr, F.L., 1995). To get the application running from any desired system, you need the following tools installed on your system:

- Python Ver 3.0.0 and above (https://www.python.org/downloads/)
- Tkinter - Python Interface (https://docs.python.org/3/library/tkinter.html)

To start up the application after installing all needed libraries, follow the instruction below and run the command in your terminal.

- unzip the file submitted
- enter into the project directory (i.e cd x4z22)
- next, run the python script inside the directory (python3 main.py)

Expect to See a Tkinter Screen prompting the user to select their start location and end location (See Figure 5).

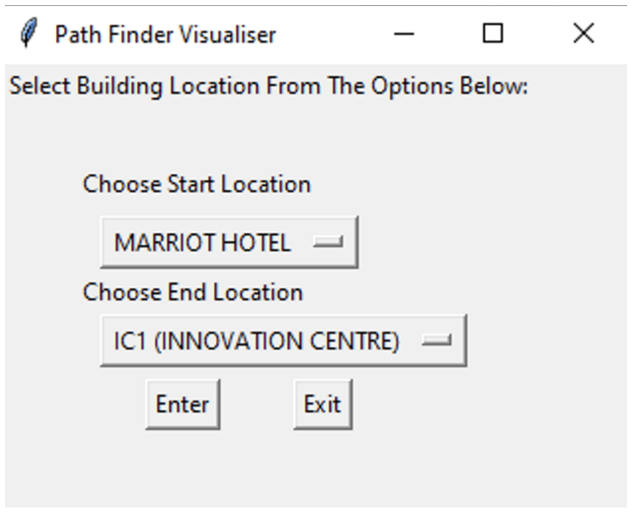


Figure 6:- Screenshot of running application

4.0 Conclusion, Evaluation, and Screenshot of The Application

Although it is the shortest pathfinding algorithm around, the A* Search Algorithm doesn't always produce the shortest path, as it heavily relies on heuristics/approximations to calculate – h.
When to use A* over other algorithms to find the shortest paths?
This is summarised below-
1) One source and One Destination-
→ A* Search Algorithm

2) One Source to All Destination –
→ BFS (For Unweighted Graphs)
→ Dijkstra (For Weighted Graphs without negative weights)
→ Bellman-Ford (For Weighted Graphs with negative weights)
3) Between every pair of nodes-
→ Floyd-Warshall
→ Johnson's Algorithm

Note: The grey cells indicate blocked cells while the red indicates the shortest route the user can take (See Figure 7).

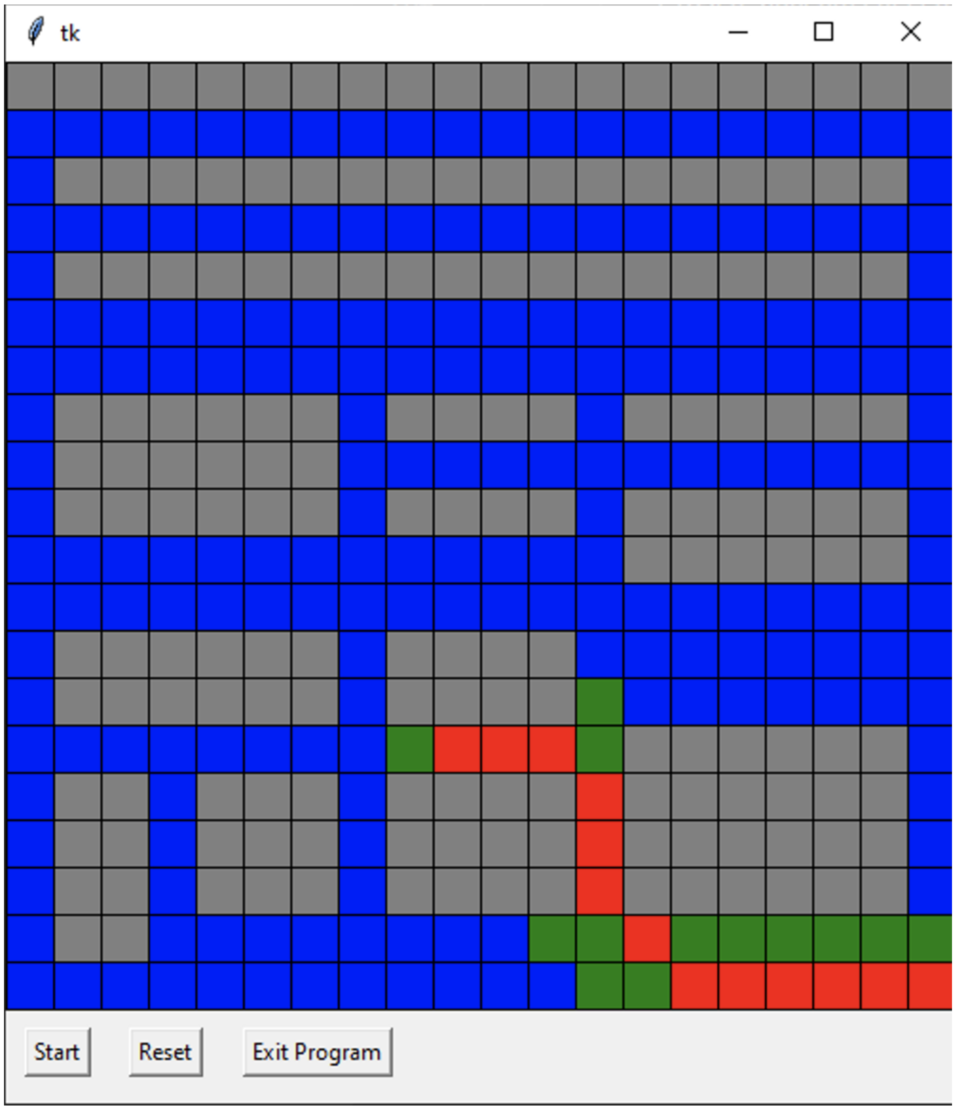


Figure 7:- Screenshot showing the shortest path between MARRIOT HOTEL and IC1 (INNOVATION CENTRE) with no obstacle set

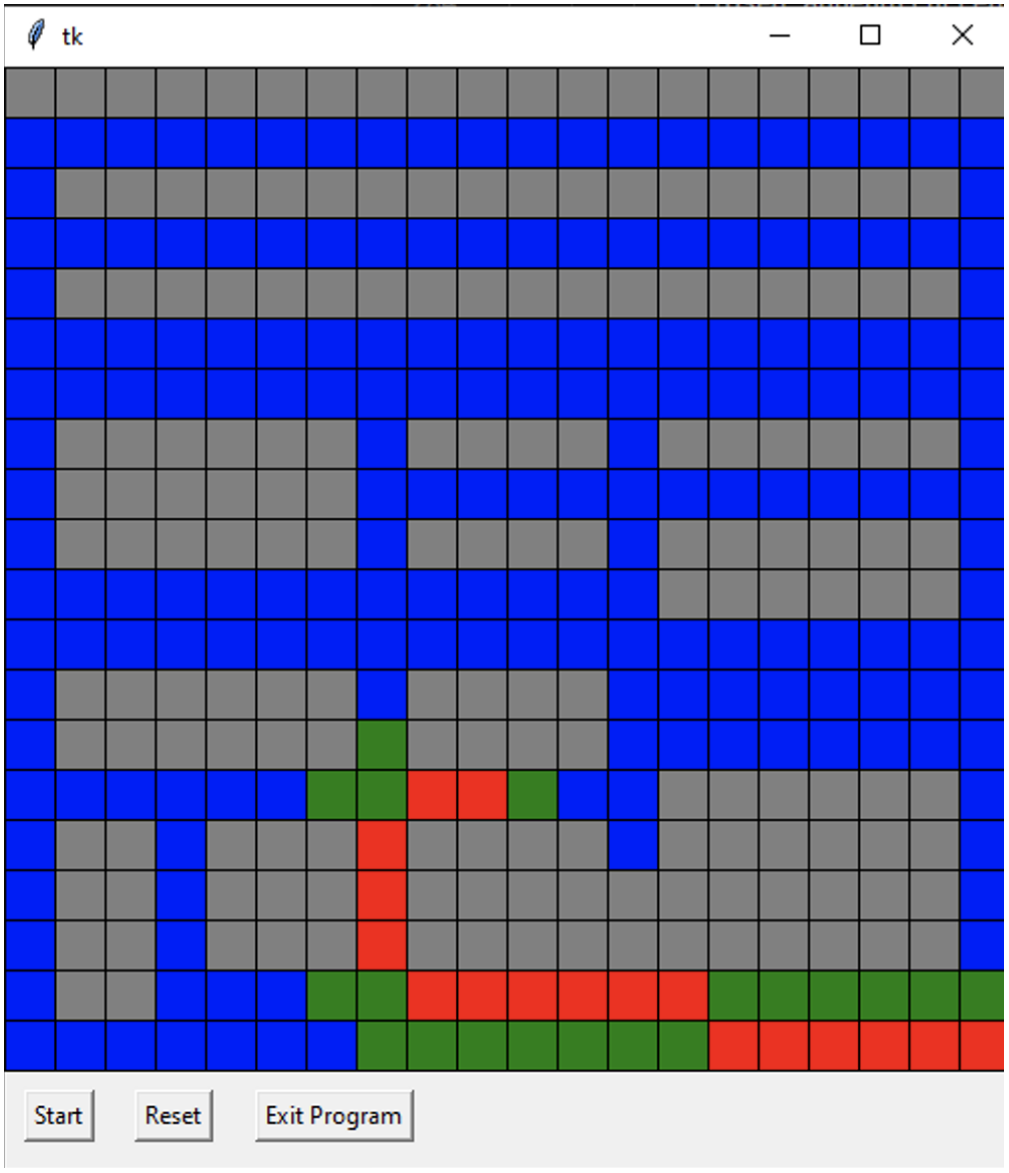


Figure 8:- Screenshot showing the shortest path between MARRIOT HOTEL and IC1 (INNOVATION CENTRE) with obstacle set (black boxes ticked)

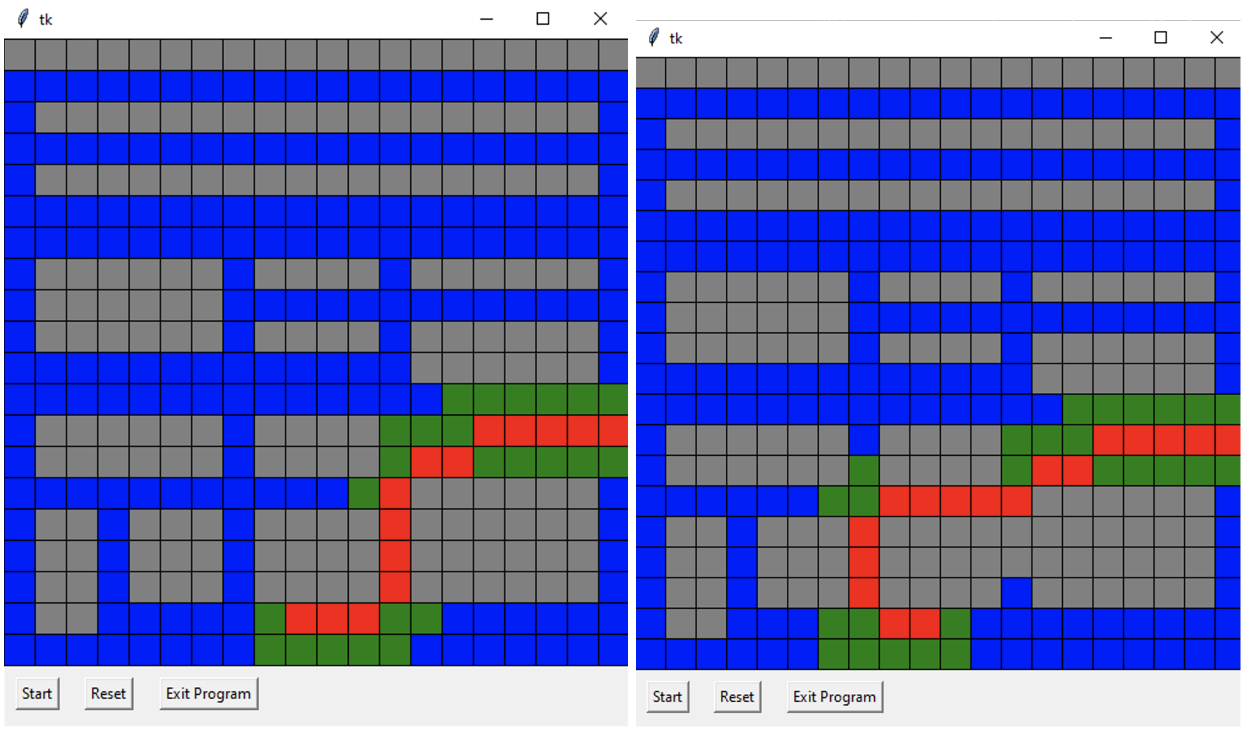
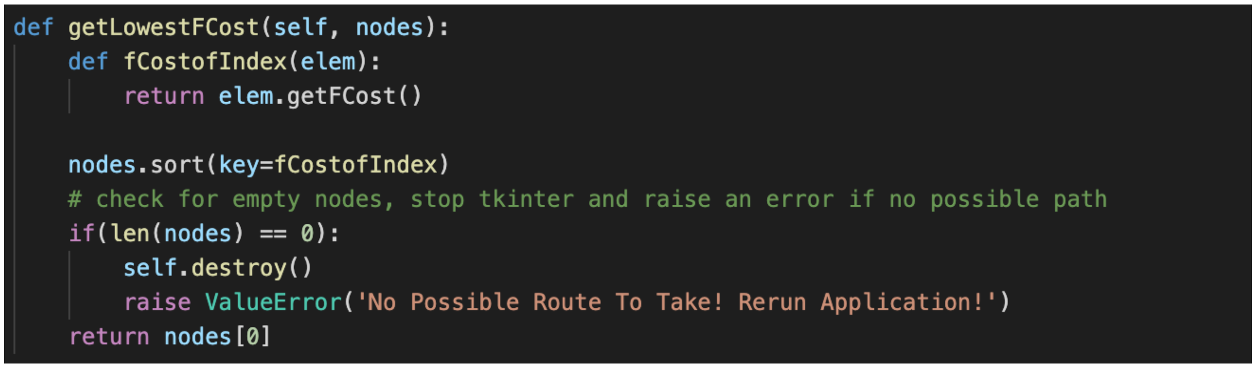


Figure 9:- Screenshot showing the shortest path between BARNES HALL and VET SCHOOL with and without obstacle set (black boxes ticked)

Note: The application is set to terminate and throw an error back to the user if there is no possible route (See Figure 10).



```python
def getLowestFCost(self, nodes):
    def fCostofIndex(elem):
        return elem.getFCost()

    nodes.sort(key=fCostofIndex)
    # check for empty nodes, stop tkinter and raise an error if no possible path
    if(len(nodes) == 0):
        self.destroy()
        raise ValueError('No Possible Route To Take! Rerun Application!')
    return nodes[0]
```

Figure 10:- Error raised due to no possible route available to the user.

5.0 References

1. Liu, X. and Gong, D., 2011, April. A comparative study of A-star algorithms for search and rescue in perfect maze. In *2011 International Conference on Electric Information and Control Engineering* (pp. 24-27). IEEE.
2. Li, D.W., Han, B.M. and Han, Y., 2007. Conversely improved A star route search algorithm. *Journal of System Simulation*, *22*.
3. Van Rossum, G. and Drake Jr, F.L., 1995. *Python tutorial* (Vol. 620). Amsterdam: Centrum voor Wiskunde en Informatica.